**CMPE273: Enterprise Distributed Systems**

**Lab 3 Assignment: Using GRAPHQL**

Devanshi Trivedi

(013530530)

https://github.com/Anuragis/CMPE273-72/tree/master/Lab3

1. INTRODUCTION

**Goals:** Understanding GrapgQL services and React JS frontend services by implementing it in HomeAway application. The aim is to create a full stack website which allows a client to search and book trips. Also, it allows users to rent their property to travelers for a specific period of time.

**Purpose of system:** Facilitate travelers to book a trip and maintain their record using MONGODB database and allowing users to post their apartment lease details.

1. SYSTEM DESIGN

This system design is a full stack web development.

Frontend: React.JS

Backend: Node.js

Database: MONGODB

GRAPHQL: It is an alternative to REST API calls. GraphQL manipulates and hands data queries.

GraphQL queries are user for getting data from database and GraphQL mutations are used to post and put data in the DB.

We have used GraphQL Appolo client which provides integration with REACT JS and Node JS and facilitates making queries to GRAPH QL.

Traveler View:

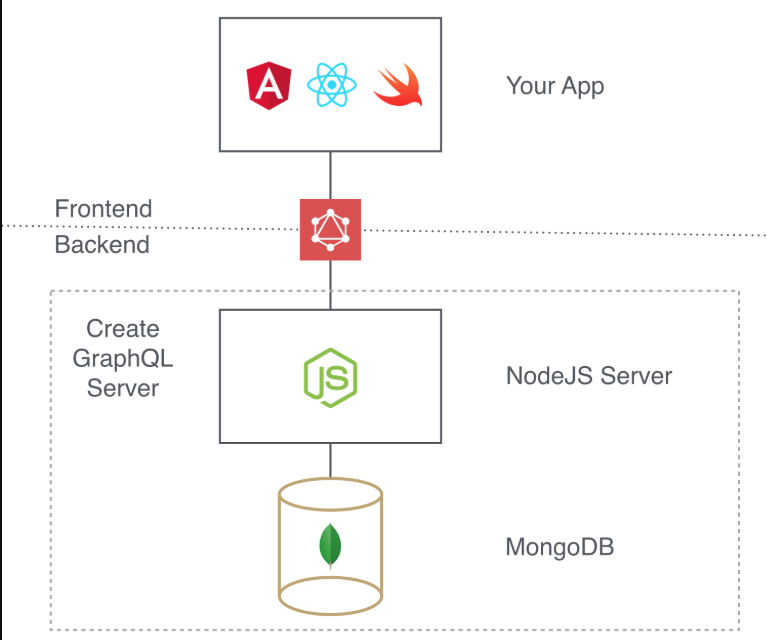
The traveler has to login to search homes to stay while travelling. The search data is sent to the backend which fires a Mongo query to get data from database which match the search results. The received data is sent to the frontend and displayed for the client.

The traveler can then book the apartment which is shown in the traveler trips page.

Owner View:

People can post their apartments for a particular time frame according to the availability. Their properties are stored in the database and shown in the owner dashboard.

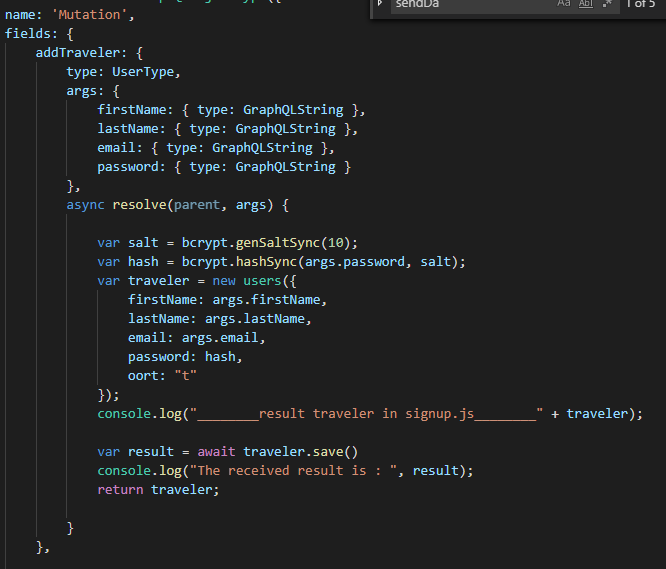
1. System Architecture

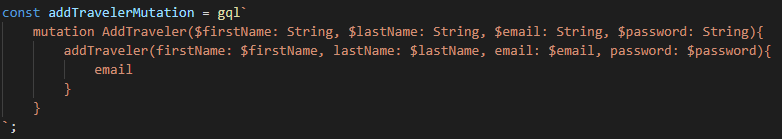


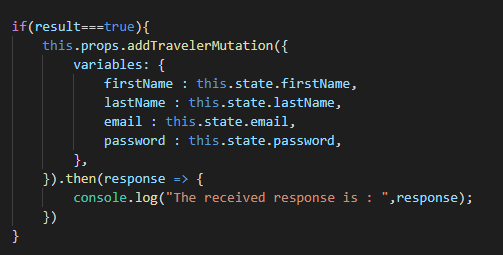
1. **GraphQL Services**

1. Traveler signup

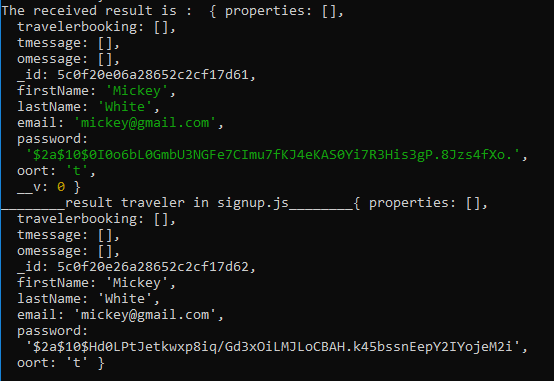
GraphQL Service



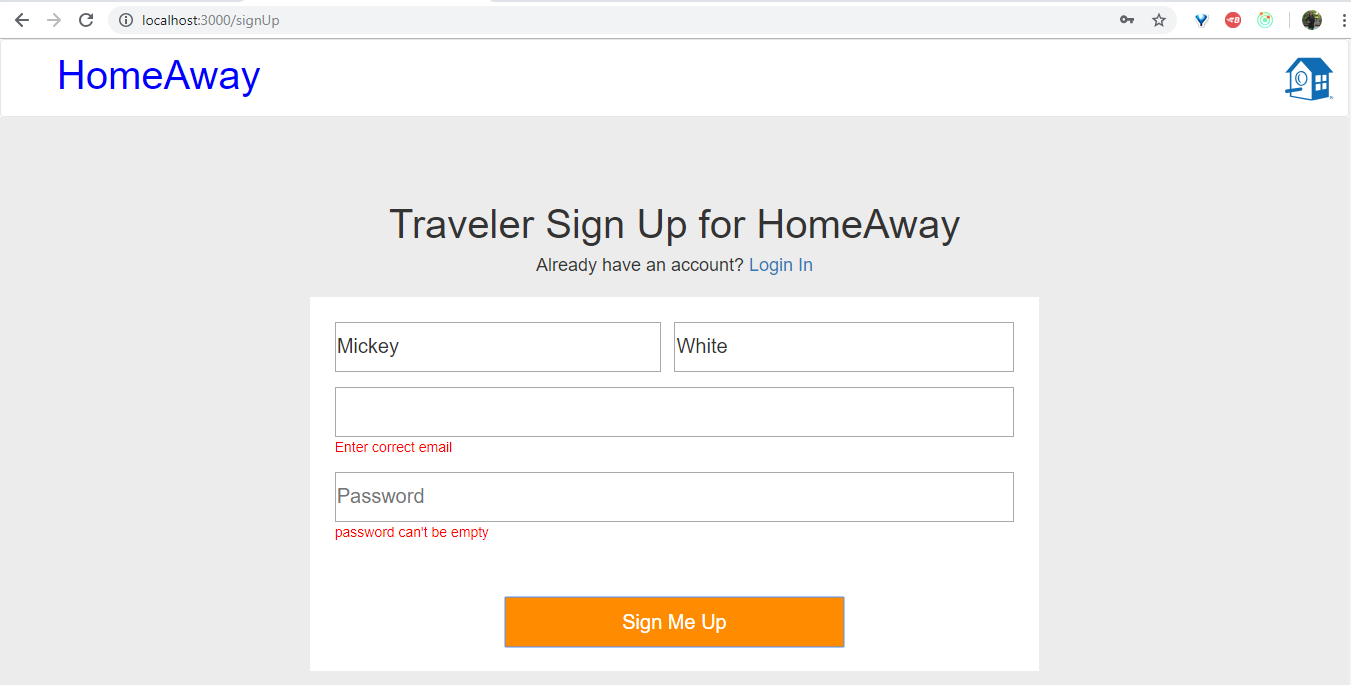




Backend



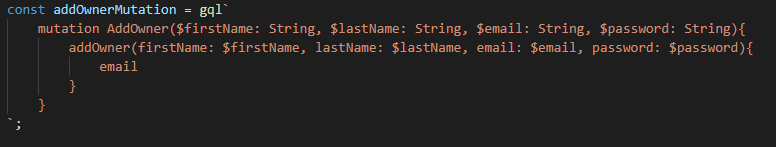
Validation

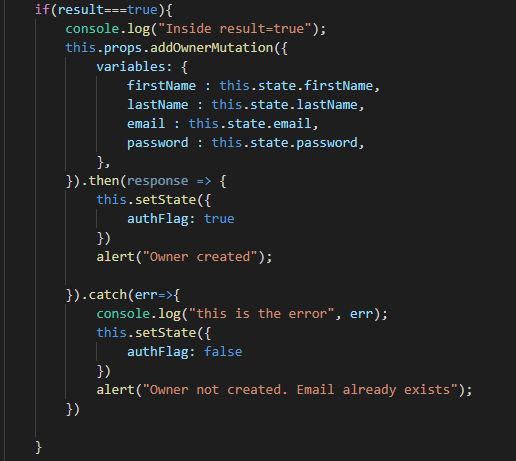


1. Owner signup

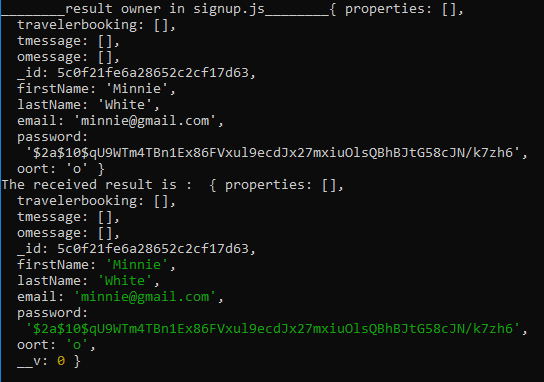
GraphQL Service



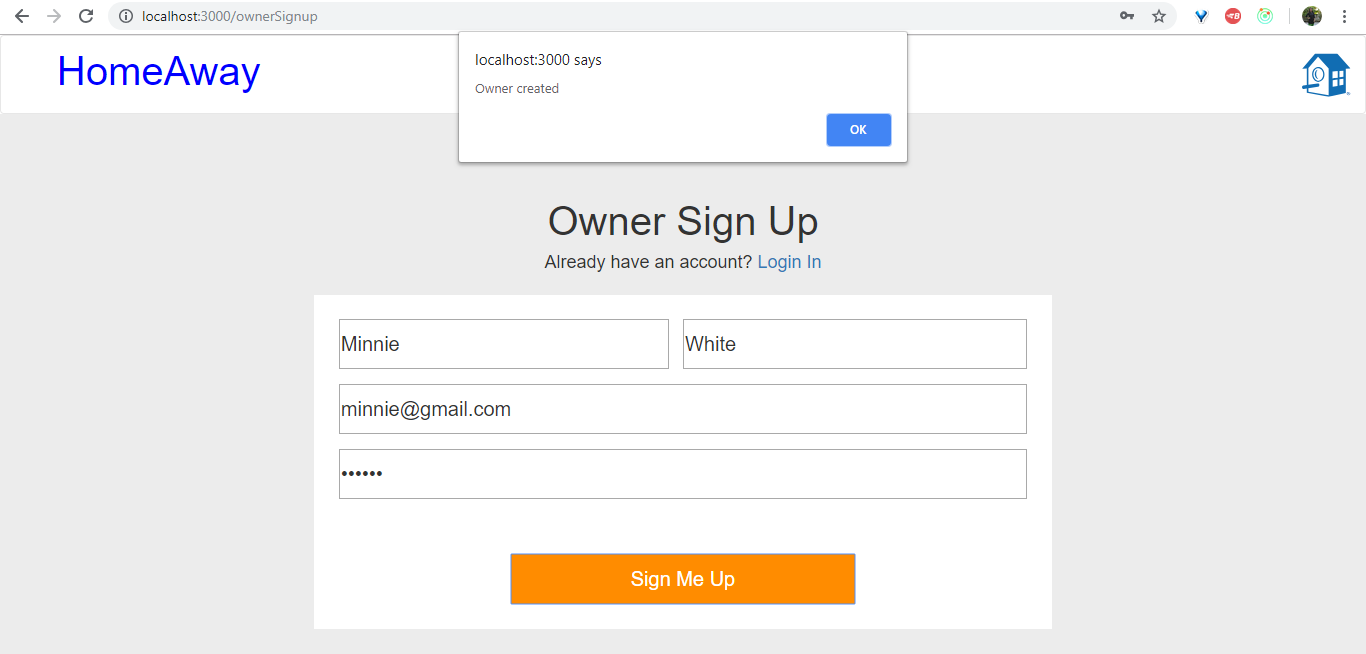




Backend

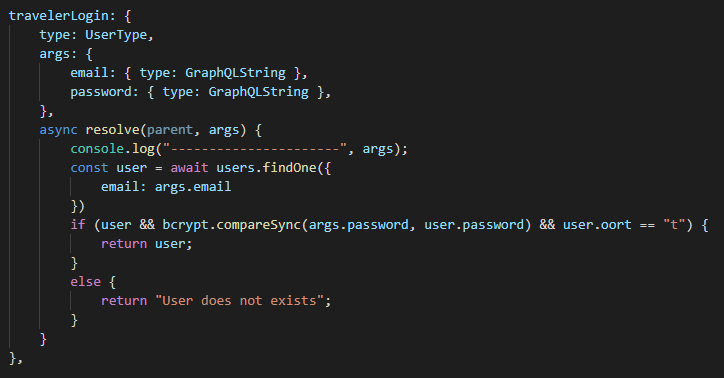


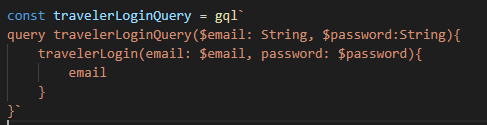
Validation



1. Traveler Login

GraphQL Schema



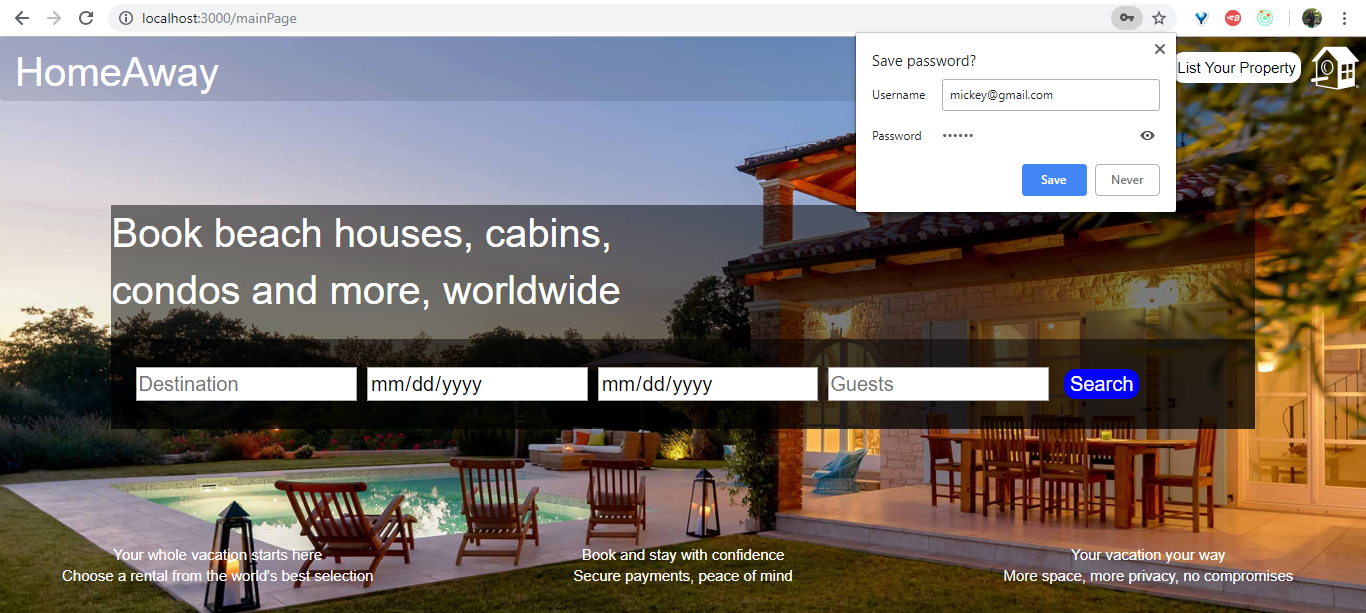




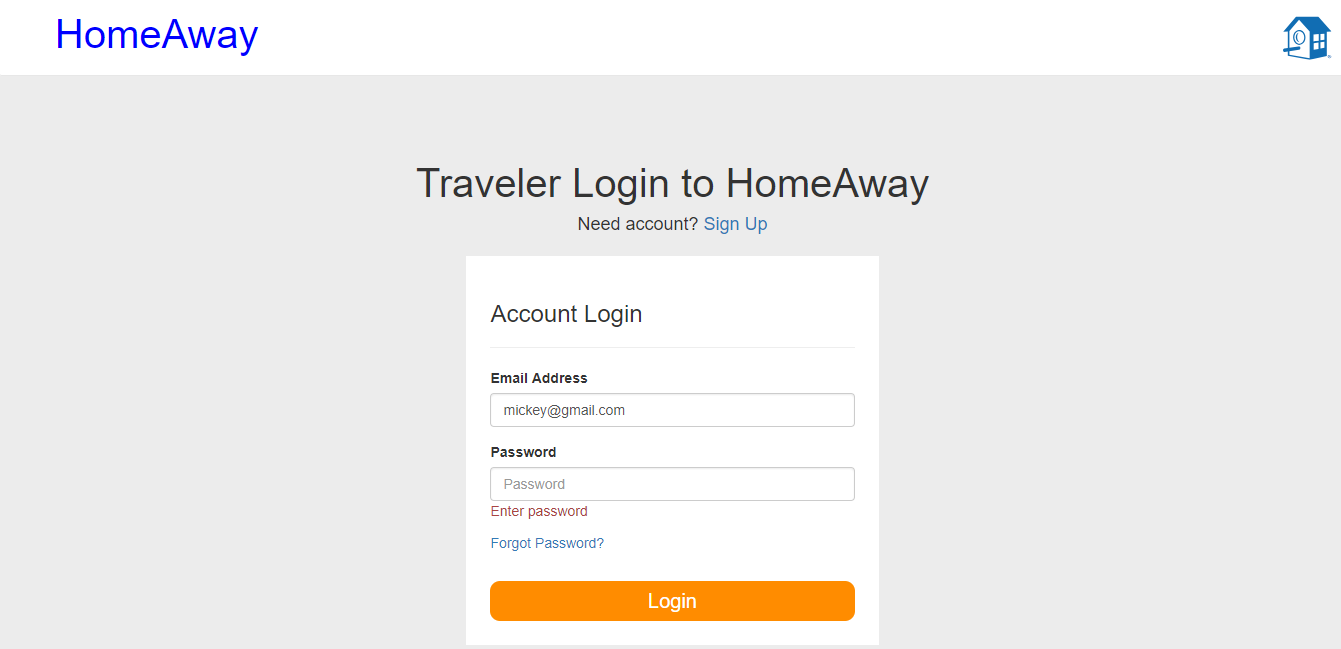
Backend



Frontend after login

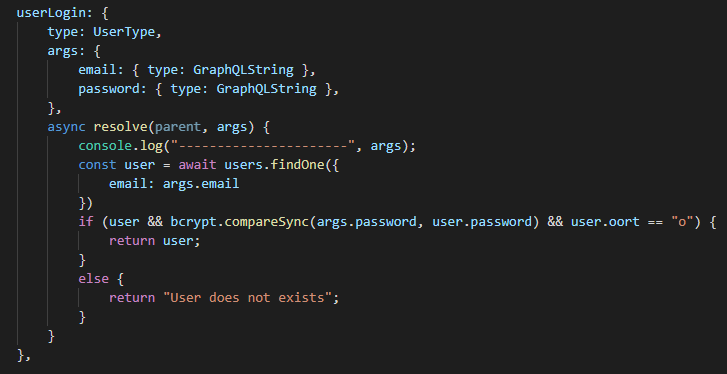


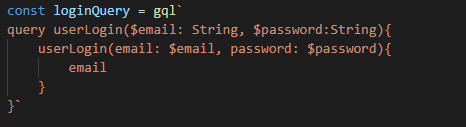
Validation



1. Owner Login

GraphQL Schema



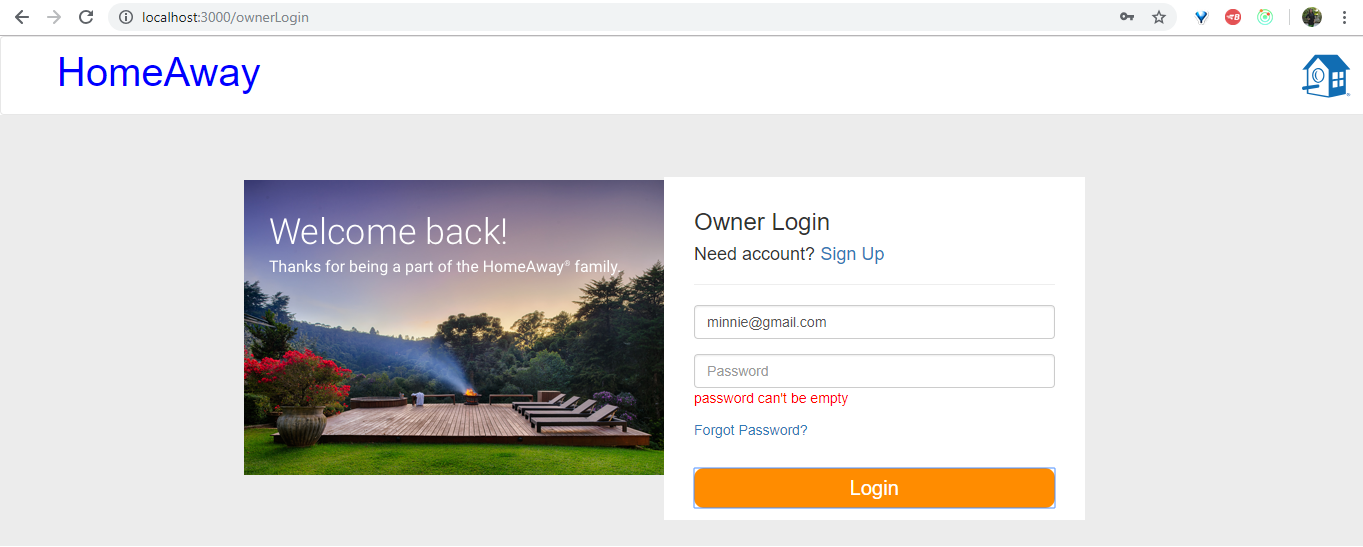




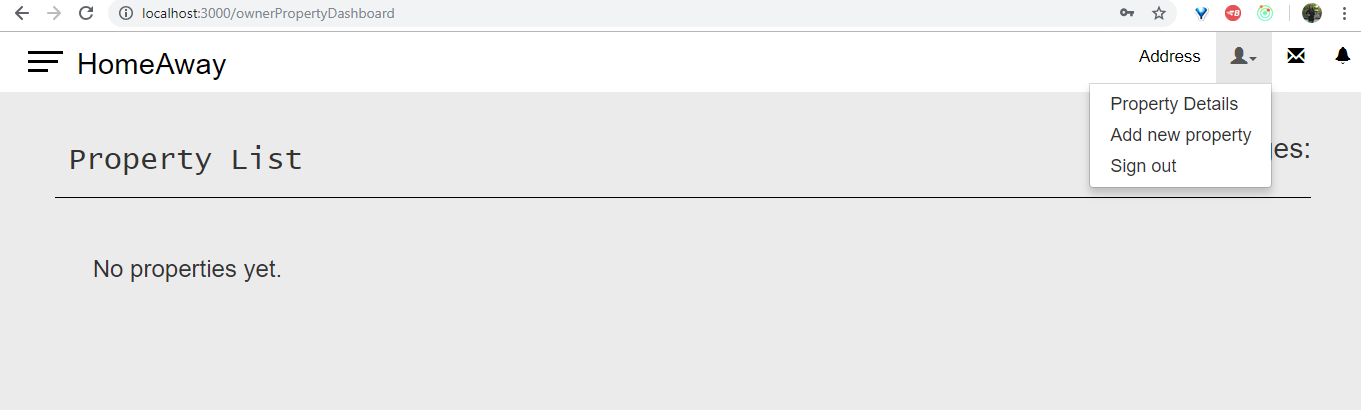
Backend



Validation

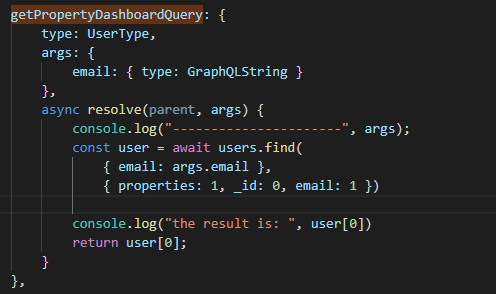


1. SignOut

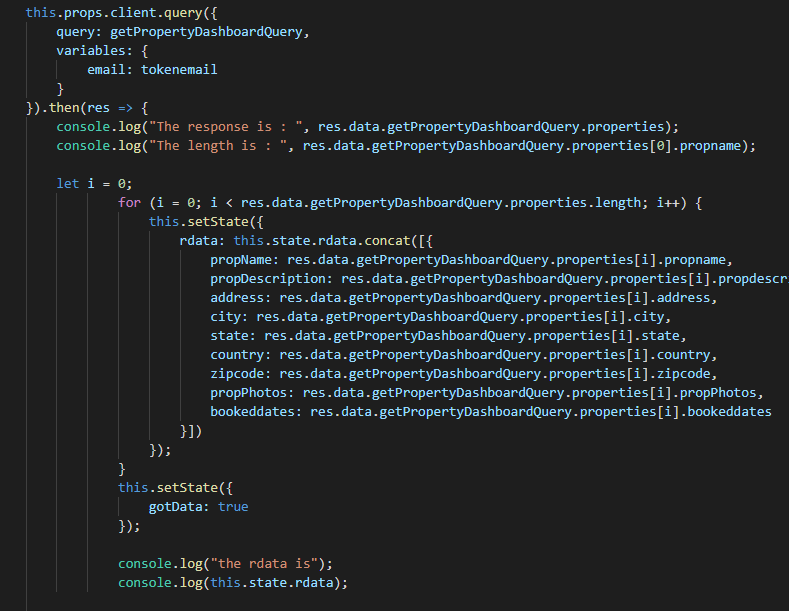


1. List Property Dashboard

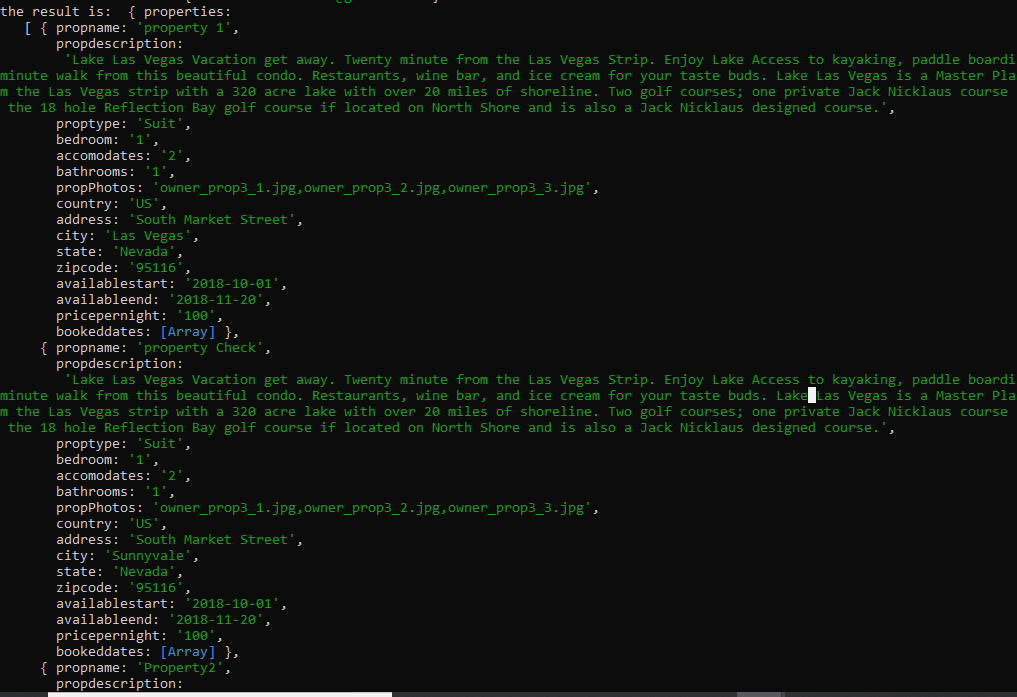
GraphQL Schema



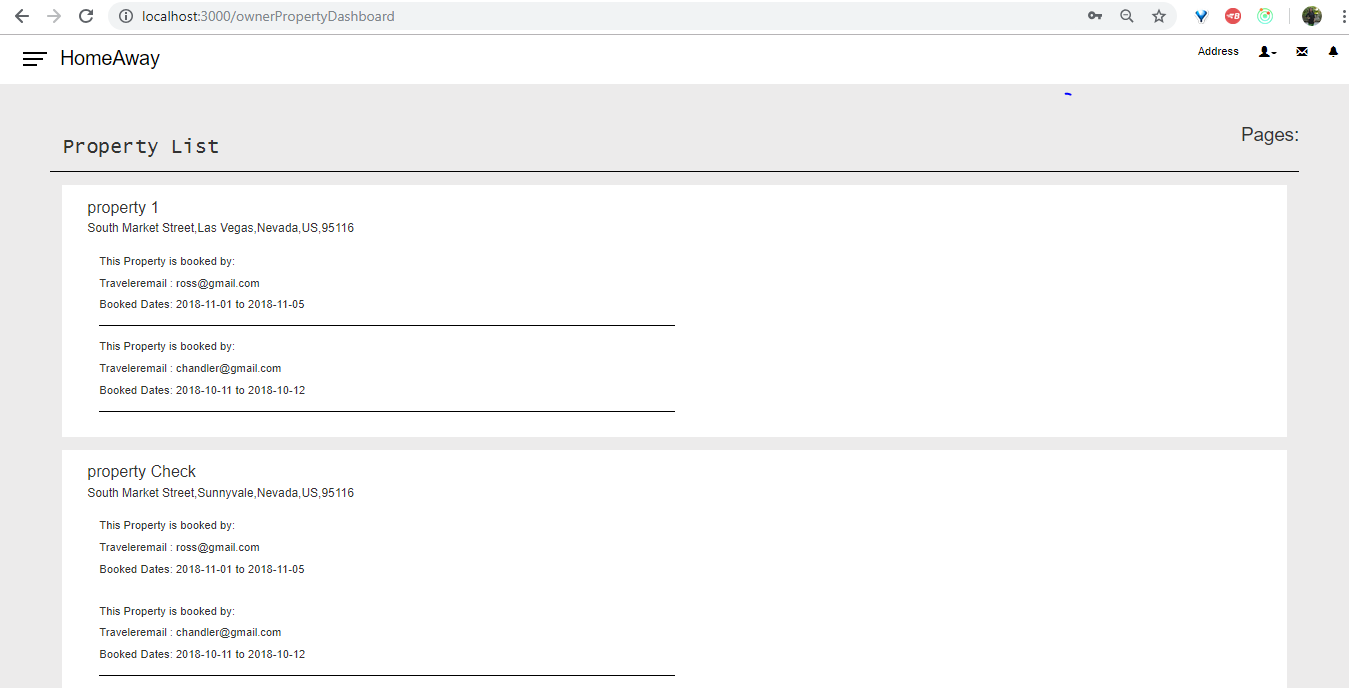




Backend

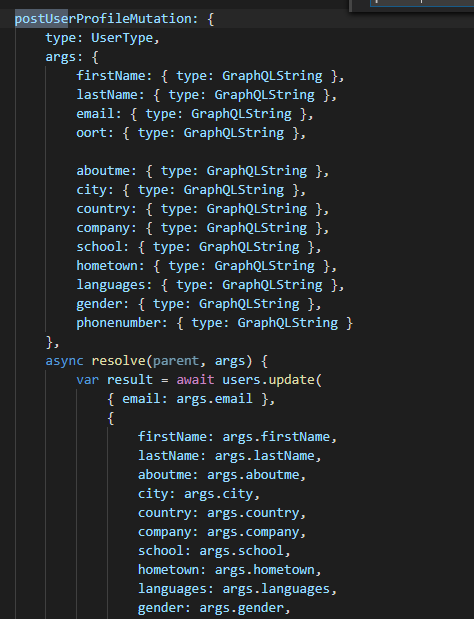


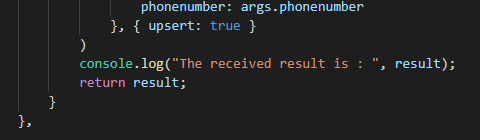
Frontend

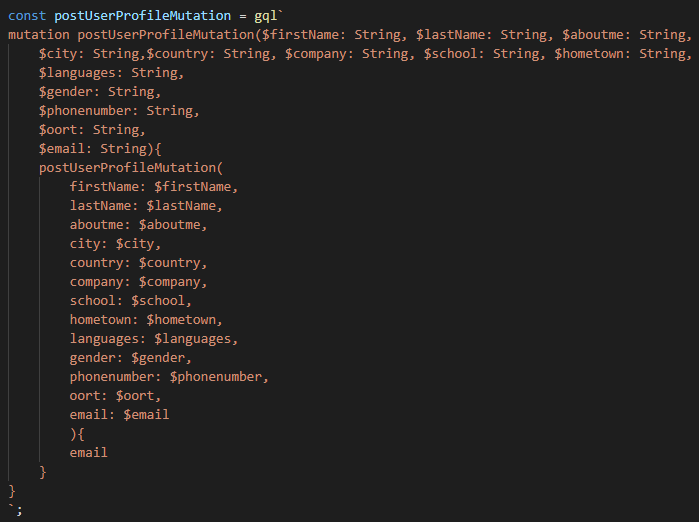


1. Update Traveler Profile

GraphQL Schema

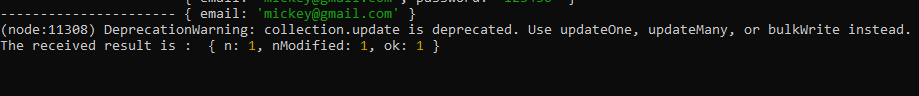




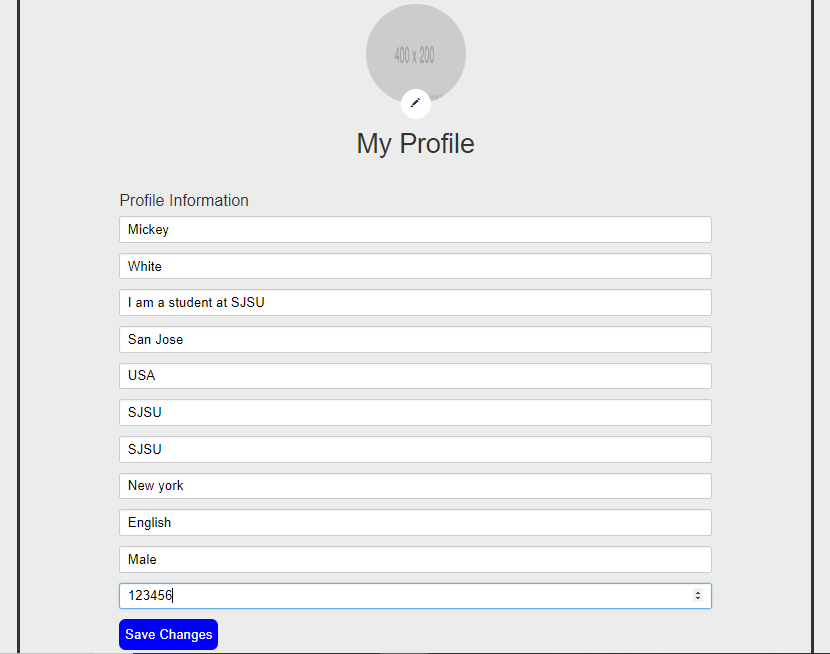


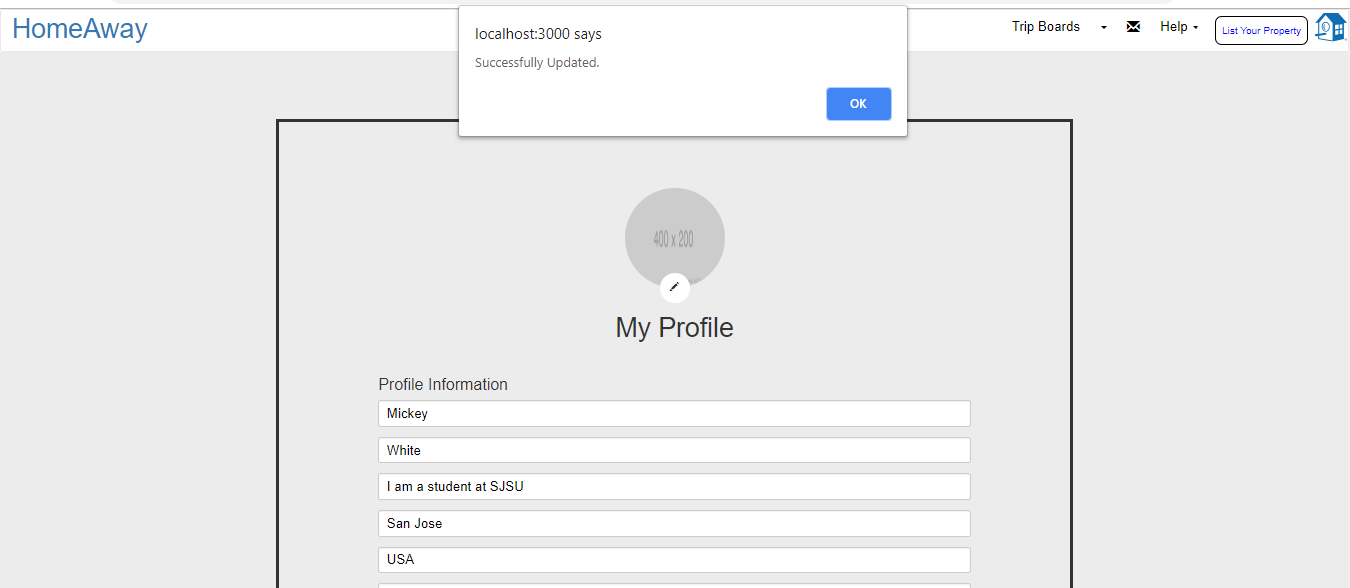


Backend



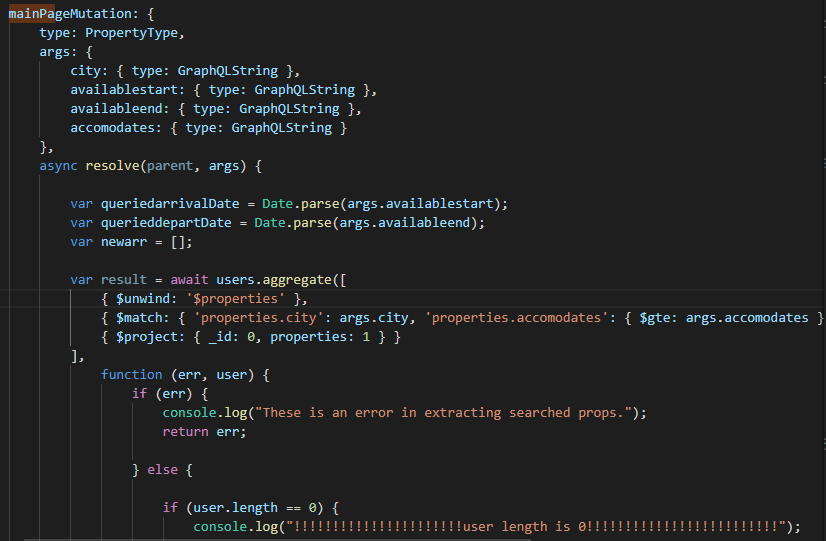
Frontend

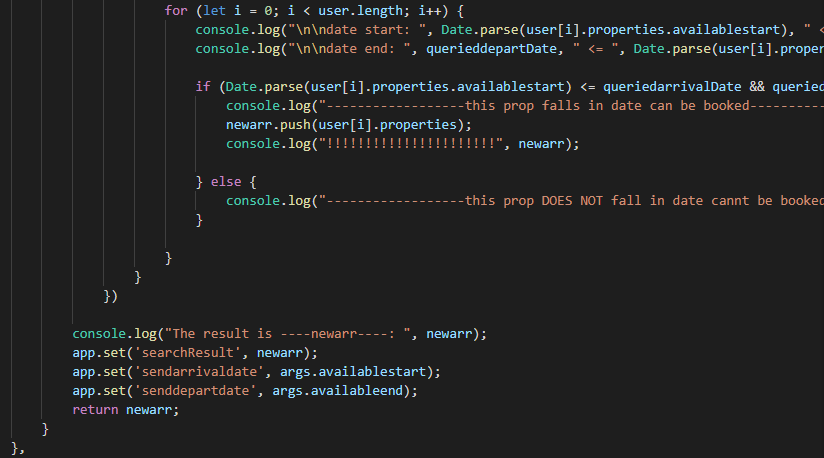


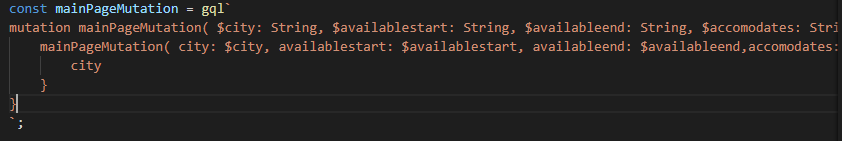


1. Search a property
2. Main Page query

GraphQL Schema

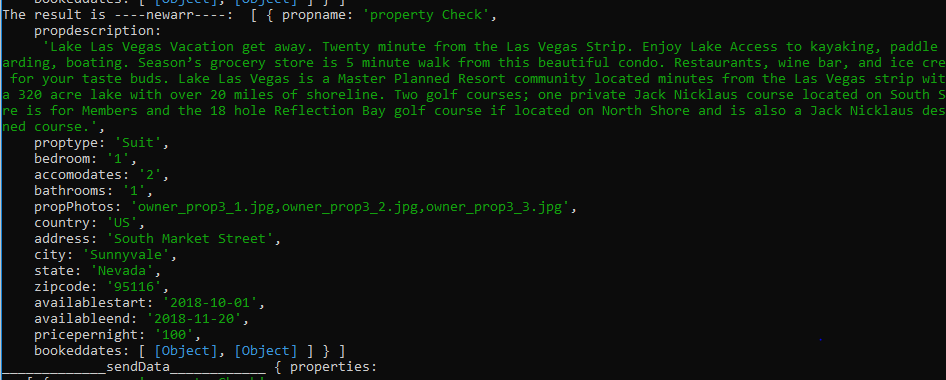




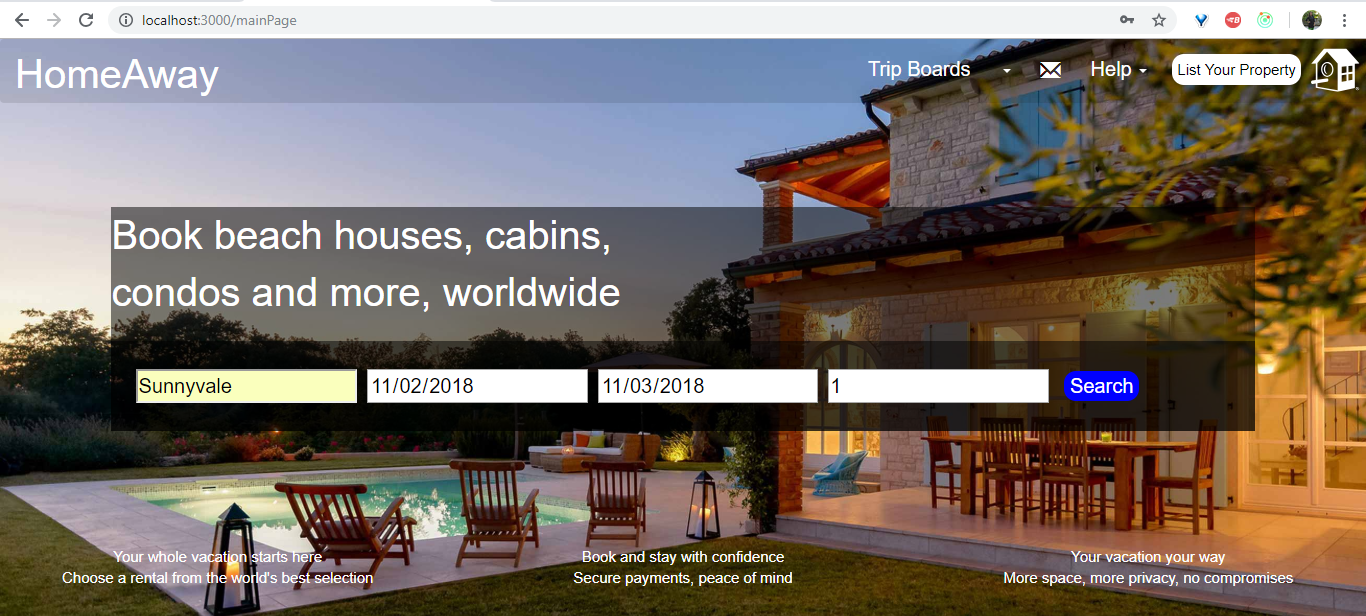




Backend

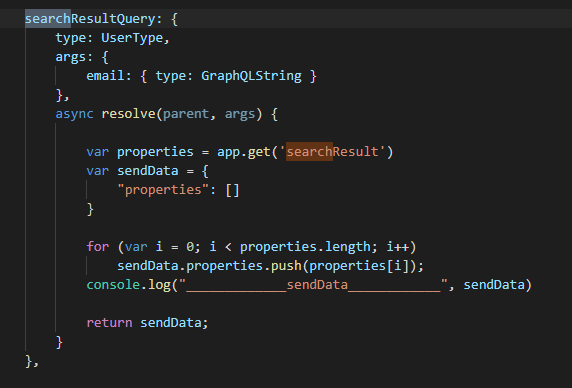


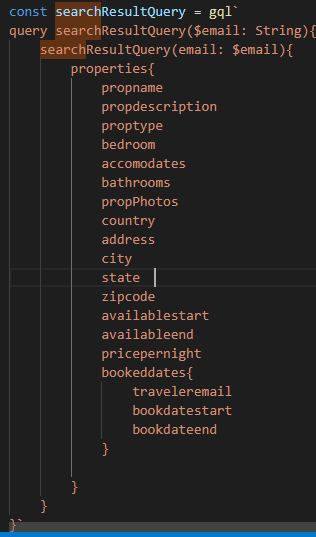
Frontend

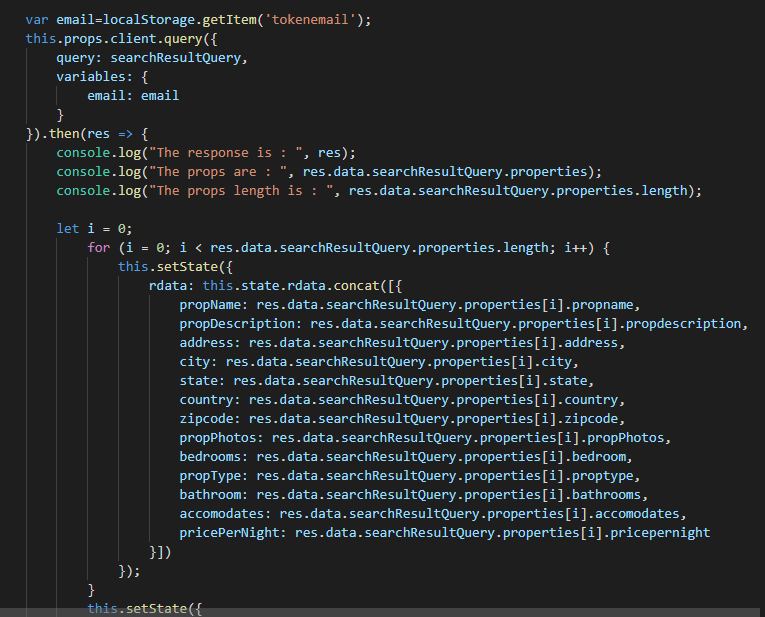


1. Get on results page

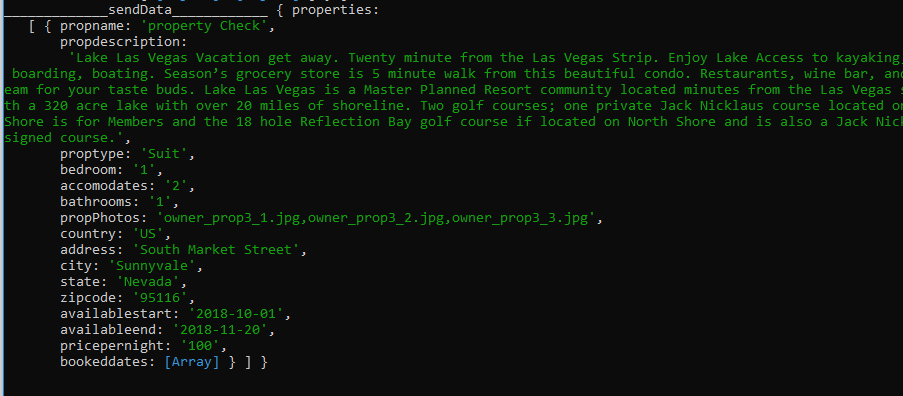
GraphQL Services



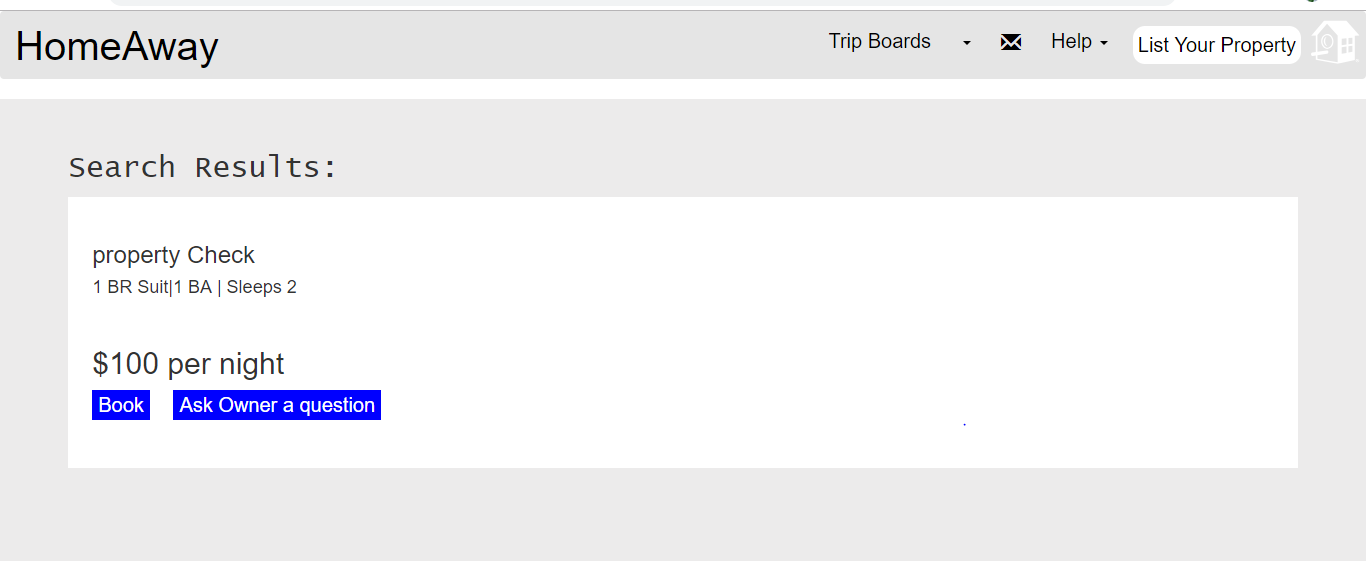




Backend

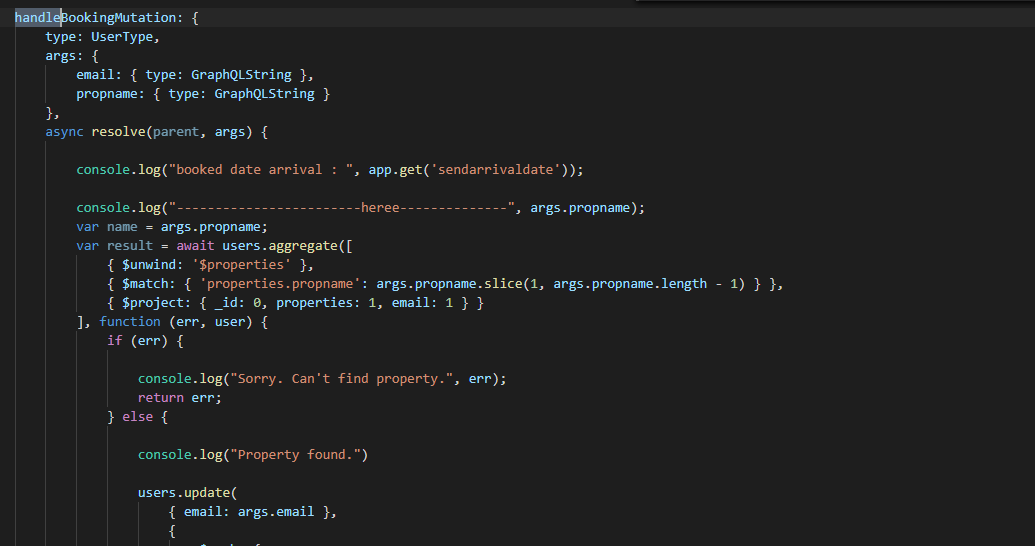


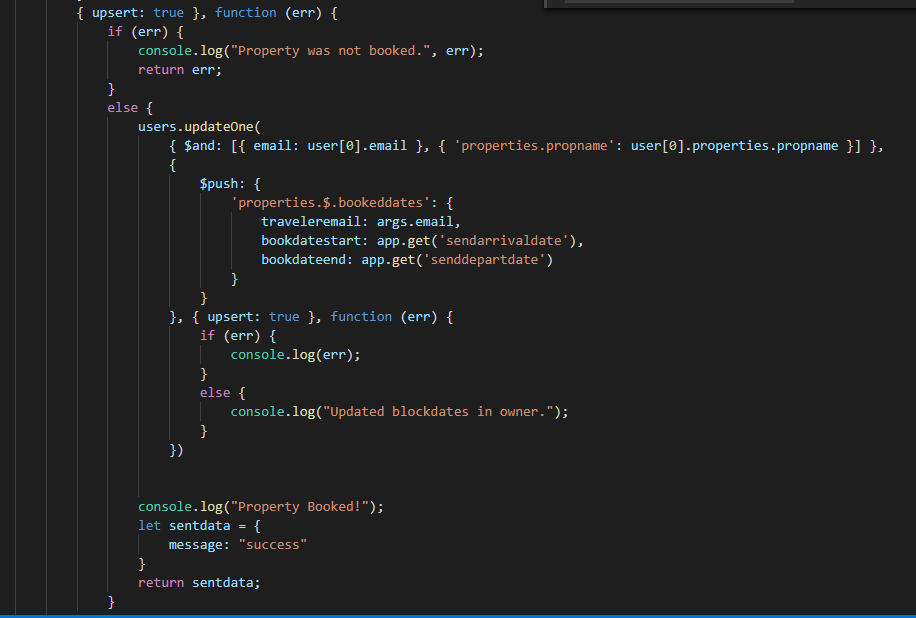
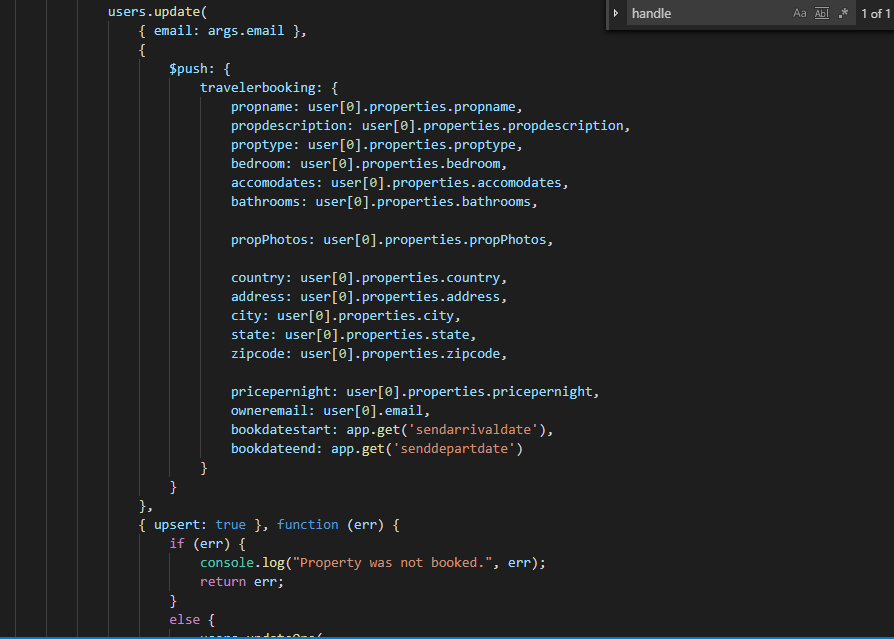
Frontend



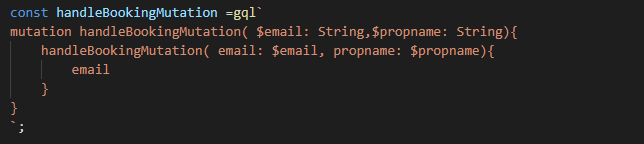
1. Make a booking

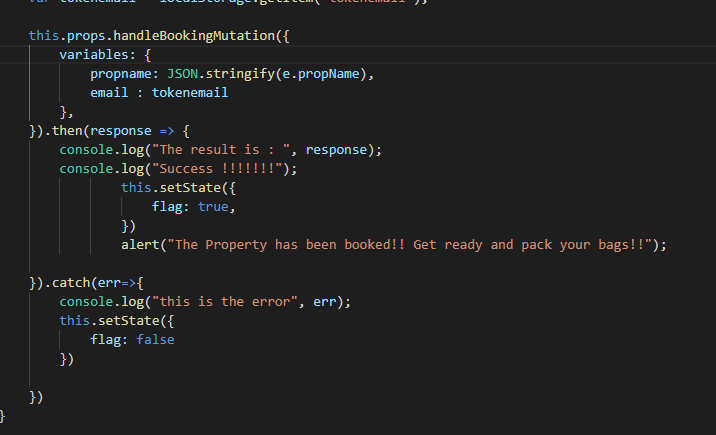
GraphQL Schema



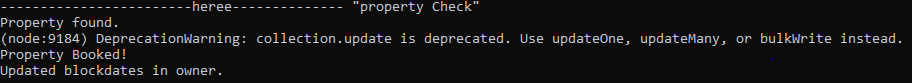




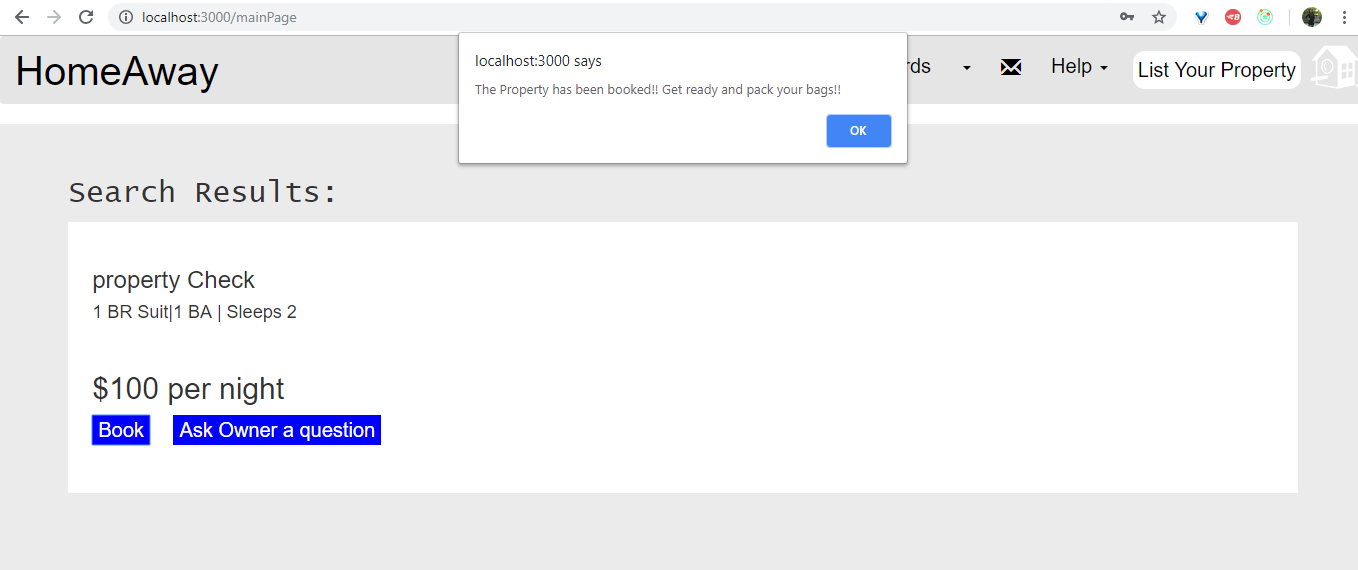




Backend

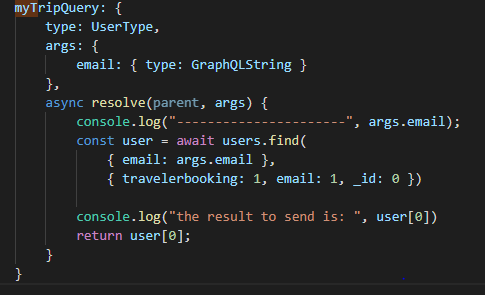


Frontend

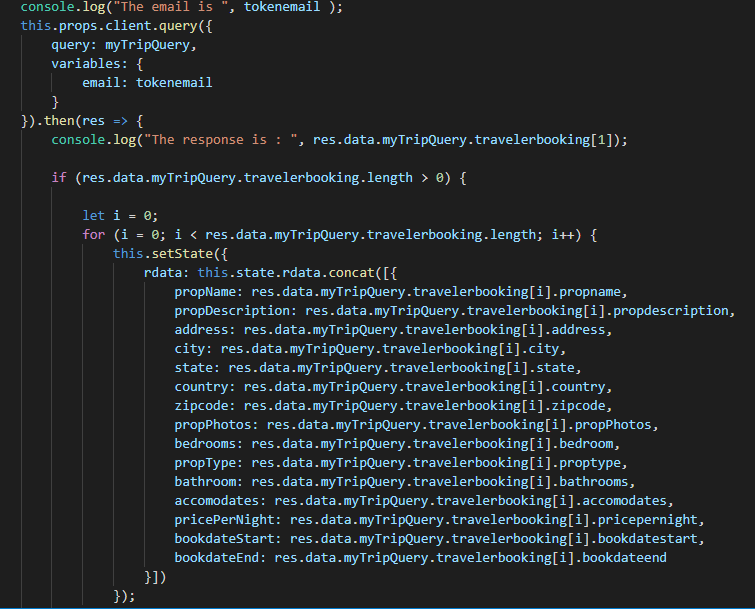


1. View Bookings

GraphQL Schema



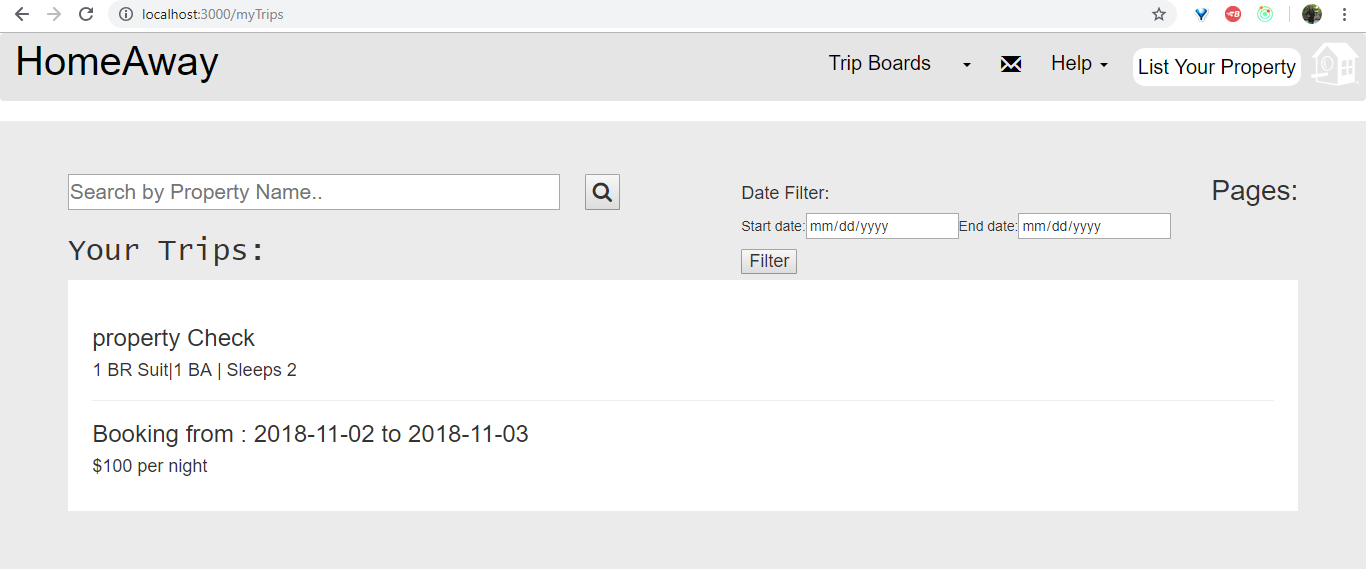




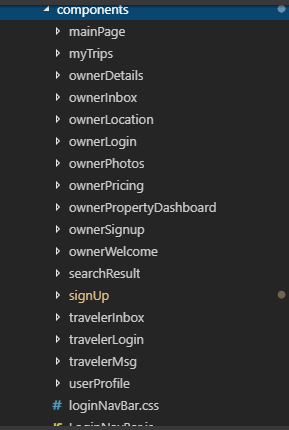
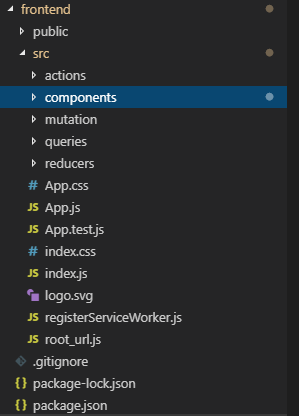
Backend

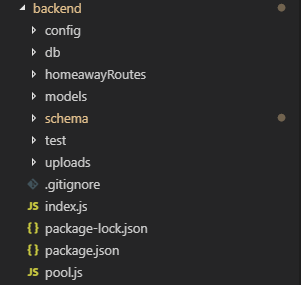


Frontend



CODE LISTING





QUESTION ANSWERS

**How will you enable multi-part data in GraphQL. Dicuss two things:**

**1. An architecture for using multi-part data in GraphQL without using any open source library from Git.**

**2. State any open source library for enabling multi-part data transfer using GraphQL with sample code. Argue why do you think that this particular library is a good fit?**

**Ans1:**

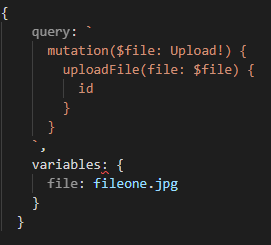
When making a post request, the data to be sent must be encoded. On the client side, whenever the data is a file to be sent, it is encoded in multipart/form-data.

One way of sending the file is to encode it to base64 and then send it with mutation data. But this is going to be very large data. Another method is to make multiple API requests. The first request can store the data and the URL of the file is sent in the net request. However, this will become asynchronous and hence is not recommended.

We use Multipart Request Spec. (Apollo server 2.0)

This makes it possible to nest files in GraphQL mutations.

Eg:



This method includes two types of specs.

One is the client-side spec which uses a key to map a file to a mutation.

The server spec determines how to parse the file and access it.

The mutation looks like this



**Ans2:**

The multipart request can be sent to the middleware synchronously or asynchronously. When passing the form data synchronously, it is temporarily stored on the API server while in async upload, the file upload streams are passed into a resolver.

The Apollo-upload server middleware is good for supporting multipart form data upload.

Firstly, you have to install graphql and Apollo-upload server.

Here the clients upload their files as a upload scalar query or in mutation variables. The resolver resolves these details for processing and storage.

This architecture has the advantage that the resolvers work asynchronously and hence is an efficient way of uploading multiple data.

**GITHUB COMMITS:**

